**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 6

**З дисципліни:** *“Дослідження операцій”*

**На тему:** *“ Розв’язування задач оптимізації на мережах за модифікованими  
алгоритмами Дейкстри та Флойда ”*

**Лектор:**

доц. каф. ПЗ

Журавчак Л. М.

**Виконав:**

ст. гр. ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

асист. каф. ПЗ

Івасько Н. М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_\_ .

Львів – 2022

**Тема роботи:** Розв’язування задач оптимізації на мережах за модифікованими алгоритмами Дейкстри та Флойда.

**Мета роботи:** Ознайомитись на практиці із основними алгоритмами  
розв’язування потокових задач, навчитись знаходити оптимальні маршрути між вершинами мережі за допомогою модифікованих алгоритмів Дейкстри та  
Флойда.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

3. Що таке орієнтована і зважена мережа?

Це мережа, що має напрямлені дуги та вагу на них.

6. На якій ідеї ґрунтується алгоритм Дейкстри та які задачі можна розв’язувати за його допомогою?

Алгоритм Дейкстри вмкористовується щоб знайти найкоротший шлях від початкової точки до всіх інших. Цим методом можна розв'язати задачі пошуку найкоротшого шляху з джерела до витоку.

9. Вкажіть, за допомогою якої модифікації можна, окрім довжини найкоротшого шляху, знайти і сам шлях у алгоритмі Дейкстри.

Додатково зберігати вектор вершин, із яких найкоротший шлях безпосередньо потрапляє у дану вершину.

**ЗАВДАННЯ**

1. Отримати індивідуальний варіант завдання.  
2. Написати програму розв’язування потокових задач за модифікованими методами Дейкстри та Флойда з Додатка до лабораторної роботи № 6.  
3. Оформити звіт про виконану роботу.  
4. Продемонструвати викладачеві результати, відповісти на запитання  
стосовно виконання роботи

**ХІД ВИКОНАННЯ**

1. Індивідуальне завдання.

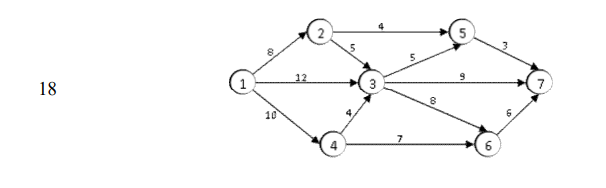


Рис. 1. Індивідуальне завдання

1. Код програми:
2. #define NOMINMAX
3. #include "RouteOptimization.h"
4. #include <json.h>
5. #include <iostream>
6. #include <algorithm>
7. #include <iomanip>
8. #include <fstream>
9. #include <sstream>
10. #include <limits>
11. #include "color.hpp"
12. void RouteOptimization::Start() {
13. CreateProblem(false, "lab6\_main.json");
14. FloydFindRoute();
15. DejkstraStart();
16. }
17. void RouteOptimization::CreateProblem(const bool fromConsole, std::string fileName) {
18. if (fromConsole) {
19. std::cout << "Enter number of peeks:";
20. int peeksNumber;
21. std::cin >> peeksNumber;
22. m\_costs.resize(peeksNumber, std::vector<double>(peeksNumber, 0));
23. std::cout << "Enter costs:";
24. for (int i = 0; i < peeksNumber; ++i) {
25. for (int j = 0; j < peeksNumber; ++j) {
26. std::cout << "c[" << i + 1 << "][" << j + 1 << "] = ";
27. std::cin >> m\_costs[i][j];
28. }
29. }
30. std::cout << "Enter the start peek:";
31. std::cin >> m\_startPeek;
32. std::cout << "Enter the end peek:";
33. std::cin >> m\_endPeek;
34. }
35. else {
36. std::ifstream jsonFile(fileName);
37. if (jsonFile.fail())
38. {
39. throw std::runtime\_error("File was not found.\n");
40. }
41. nlohmann::json parsedJson = nlohmann::json::parse(jsonFile);
42. const auto& parsedJsonCosts = parsedJson.at("Costs");
43. const auto peeksCount = parsedJson.at("PeeksCount").get<int>();
44. m\_costs.resize(peeksCount, std::vector<double>(peeksCount, 0));
45. for (int i = 0; i < peeksCount; ++i) {
46. for (int j = 0; j < peeksCount; ++j) {
47. m\_costs[i][j] = parsedJsonCosts[i][j].get<double>();
48. }
49. }
50. m\_startPeek = parsedJson.at("StartPeek").get<int>();
51. m\_endPeek = parsedJson.at("EndPeek").get<int>();
52. }
53. }
54. void RouteOptimization::PrintFloyd(const int iteration, std::vector<std::pair<int, int>> underlineIndices) {
55. std::cout << "\nIteration " << iteration << ":\n";
56. for (int rowIndex = 0; rowIndex < m\_costs.size(); ++rowIndex) {
57. if (rowIndex == 0) {
58. std::cout << std::setw(5) << std::left << "D: [";
59. }
60. else {
61. std::cout << std::setw(5) << " [";
62. }
63. for (int colIndex = 0; colIndex < m\_costs.size(); ++colIndex) {
64. auto underline = std::find\_if(underlineIndices.begin(), underlineIndices.end(),
65. [=](const auto& indices)
66. {
67. return (rowIndex == indices.first) && (colIndex == indices.second);
68. });
69. if (underline != underlineIndices.end()) {
70. std::ostringstream ss;
71. ss << std::setprecision(1) << std::left << std::fixed << m\_floyd\_d[rowIndex][colIndex];
72. std::cout << std::setw(10) << dye::red((m\_floyd\_d[rowIndex][colIndex] == std::numeric\_limits<double>::max()) ? "-" : ss.str());
73. }
74. else {
75. std::ostringstream ss;
76. ss << std::setprecision(1) << std::left << std::fixed << m\_floyd\_d[rowIndex][colIndex];
77. std::cout << std::setw(10) << ((m\_floyd\_d[rowIndex][colIndex] == std::numeric\_limits<double>::max()) ? "-" : ss.str());
78. }
79. if (colIndex < m\_costs.size() - 1) {
80. std::cout << ", ";
81. }
82. }
83. std::cout << std::setw(10) << std::left << " ] ";
84. if (rowIndex == 0) {
85. std::cout << std::setw(5) << std::left << "R: [";
86. }
87. else {
88. std::cout << std::setw(5) << "";
89. }
90. for (int colIndex = 0; colIndex < m\_costs.size(); ++colIndex) {
91. auto underline = std::find\_if(underlineIndices.begin(), underlineIndices.end(),
92. [=](const auto& indices)
93. {
94. return (rowIndex == indices.first) && (colIndex == indices.second);
95. });
96. if (underline != underlineIndices.end()) {
97. std::ostringstream ss;
98. ss << std::setw(10) << std::left << m\_floyd\_r[rowIndex][colIndex];
99. std::cout << dye::red(ss.str());
100. }
101. else {
102. std::cout << std::setw(10) << std::left << m\_floyd\_r[rowIndex][colIndex];
103. }
104. if (colIndex < m\_costs.size() - 1) {
105. std::cout << ", ";
106. }
107. }
108. std::cout << std::left << " ] \n";
109. }
110. }
111. void RouteOptimization::FloydCreateTables() {
112. m\_floyd\_d = m\_costs;
113. m\_floyd\_r.resize(m\_costs.size(), std::vector<int>(m\_costs.size(), 0));
114. for (int i = 0; i < m\_costs.size(); ++i) {
115. for (int j = 0; j < m\_costs.size(); ++j) {
116. if (i == j) {
117. m\_floyd\_d[i][j] = 0;
118. m\_floyd\_r[i][j] = 0;
119. continue;
120. }
121. if (m\_floyd\_d[i][j] == 0) {
122. m\_floyd\_d[i][j] = std::numeric\_limits<double>::max();
123. }
124. m\_floyd\_r[i][j] = i + 1;
125. }
126. }
127. PrintFloyd(0);
128. }
129. void RouteOptimization::FloydGoToNextTables(const int basisPeek) {
130. if (basisPeek >= m\_floyd\_d.size()) {
131. return;
132. }
133. std::vector<int> unusedRows;
134. std::vector<int> unusedCols;
135. std::vector<std::pair<int, int>> underlineIndices;
136. for (int rowIndex = 0; rowIndex < m\_costs.size(); ++rowIndex) {
137. if (m\_floyd\_d[rowIndex][basisPeek] == std::numeric\_limits<double>::max()) {
138. unusedRows.push\_back(rowIndex);
139. }
140. }
141. for (int colIndex = 0; colIndex < m\_costs.size(); ++colIndex) {
142. if (m\_floyd\_d[basisPeek][colIndex] == std::numeric\_limits<double>::max()) {
143. unusedCols.push\_back(colIndex);
144. }
145. }
146. for (int rowIndex = 0; rowIndex < m\_costs.size(); ++rowIndex) {
147. for (int colIndex = 0; colIndex < m\_costs.size(); ++colIndex) {
148. if (rowIndex == colIndex || rowIndex == basisPeek || colIndex == basisPeek
149. || (std::find(unusedRows.begin(), unusedRows.end(), rowIndex) != unusedRows.end())
150. || (std::find(unusedCols.begin(), unusedCols.end(), colIndex) != unusedCols.end())) {
151. continue;
152. }
153. auto new\_d = std::min(m\_floyd\_d[rowIndex][colIndex], (m\_floyd\_d[rowIndex][basisPeek] + m\_floyd\_d[basisPeek][colIndex]));
154. if (new\_d < m\_floyd\_d[rowIndex][colIndex]) {
155. m\_floyd\_d[rowIndex][colIndex] = new\_d;
156. m\_floyd\_r[rowIndex][colIndex] = basisPeek + 1;
157. underlineIndices.push\_back(std::pair<int, int>(rowIndex, colIndex));
158. }
159. }
160. }
161. PrintFloyd(basisPeek+1, underlineIndices);
162. FloydGoToNextTables(basisPeek + 1);
163. }
164. void RouteOptimization::FloydFindRoute() {
165. FloydCreateTables();
166. FloydGoToNextTables(0);
167. std::cout << "\nRoute from " << m\_startPeek << " to " << m\_endPeek << " :\n";
168. int i = m\_startPeek - 1;
169. int j = m\_endPeek - 1;
170. std::cout << "d = " << m\_floyd\_d[i][j] << "\n";
171. std::vector<int> route;
172. while (m\_floyd\_r[i][j] != m\_startPeek) {
173. route.push\_back(m\_floyd\_r[i][j]);
174. j = m\_floyd\_r[i][j] - 1;
175. }
176. std::cout << "Route: " << m\_startPeek << " -> ";
177. for (int m = route.size() - 1; m >= 0; --m) {
178. std::cout << route[m] << " -> ";
179. }
180. std::cout << m\_endPeek << "\n";
181. }
182. void RouteOptimization::PrintDejkstra() {
183. std::cout << "\n";
184. for (int i = 0; i < m\_dejkstra\_tree.size(); ++i) {
185. std::cout << "| " << std::setw(10) << std::left << i + 1;
186. }
187. std::cout << "|\n";
188. for (int i = 0; i < m\_dejkstra\_tree.size(); ++i) {
189. std::ostringstream ss;
190. ss << std::setprecision(1) << std::left << std::fixed << m\_dejkstra\_tree[i].first;
191. std::cout << "| " << std::setw(10) << ((m\_dejkstra\_tree[i].first == std::numeric\_limits<double>::max()) ? "-" : ss.str());
192. }
193. std::cout << "|\n";
194. for (int i = 0; i < m\_dejkstra\_tree.size(); ++i) {
195. std::cout << "| " << std::setw(10) << std::left << m\_dejkstra\_tree[i].second + 1;
196. }
197. std::cout << "|\n";
198. }
199. void RouteOptimization::DejkstraCreate() {
200. m\_dejkstra\_tree.resize(m\_costs.size(), std::pair<double, int>(std::numeric\_limits<double>::max(), 0));
201. m\_dejkstra\_tree[0].first = 0;
202. }
203. void RouteOptimization::DeikstraFindRoute() {
204. std::cout << "\nRoute from " << m\_startPeek << " to " << m\_endPeek << " :\n";
205. int i = m\_startPeek - 1;
206. int j = m\_endPeek - 1;
207. std::cout << "d = " << m\_dejkstra\_tree[j].first << "\n";
208. std::vector<int> route;
209. while (m\_dejkstra\_tree[j].second != i) {
210. route.push\_back(m\_dejkstra\_tree[j].second + 1);
211. j = m\_dejkstra\_tree[j].second;
212. }
213. std::cout << "Route: " << m\_startPeek << " -> ";
214. for (int m = route.size() - 1; m >= 0; --m) {
215. std::cout << route[m] << " -> ";
216. }
217. std::cout << m\_endPeek << "\n";
218. }
219. void RouteOptimization::DejkstraStart() {
220. DejkstraCreate();
221. DejkstraGoToNextPeek(0);
222. PrintDejkstra();
223. DeikstraFindRoute();
224. }
225. void RouteOptimization::DejkstraGoToNextPeek(const int looked\_peek) {
226. m\_looked\_peeks.push\_back(looked\_peek);
227. auto min = std::numeric\_limits<double>::max();
228. auto min\_index = -1;
229. for (int i = 0; i < m\_dejkstra\_tree.size(); ++i) {
230. if (std::find(m\_looked\_peeks.begin(), m\_looked\_peeks.end(), i) != m\_looked\_peeks.end()) {
231. continue;
232. }
233. auto new\_d = std::min(m\_dejkstra\_tree[i].first,
234. ((m\_costs[looked\_peek][i] == 0) ? std::numeric\_limits<double>::max() : m\_dejkstra\_tree[looked\_peek].first + m\_costs[looked\_peek][i]));
235. if (new\_d < m\_dejkstra\_tree[i].first) {
236. m\_dejkstra\_tree[i].first = new\_d;
237. m\_dejkstra\_tree[i].second = looked\_peek;
238. }
239. if (min > m\_dejkstra\_tree[i].first) {
240. min = m\_dejkstra\_tree[i].first;
241. min\_index = i;
242. }
243. }
244. std::cout << std::endl;
245. PrintDejkstra();
246. std::cout << "\ny = " << min\_index + 1 << std::endl;
247. if (min\_index == -1 || min\_index == m\_endPeek - 1) {
248. std::cout << "\nEnd.\n";
249. return;
250. }
251. DejkstraGoToNextPeek(min\_index);
252. }

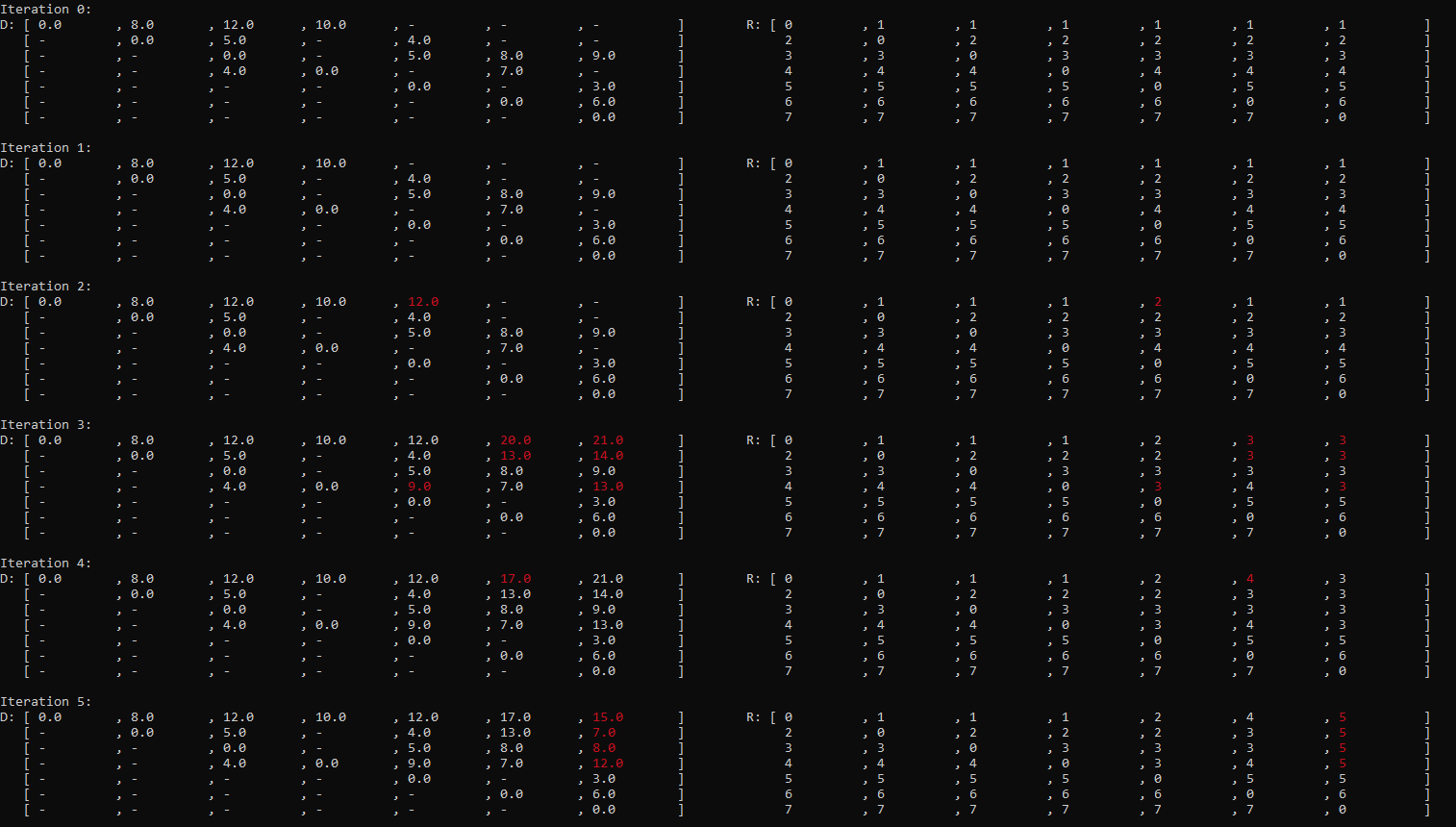


Рис. 2. Результати програми

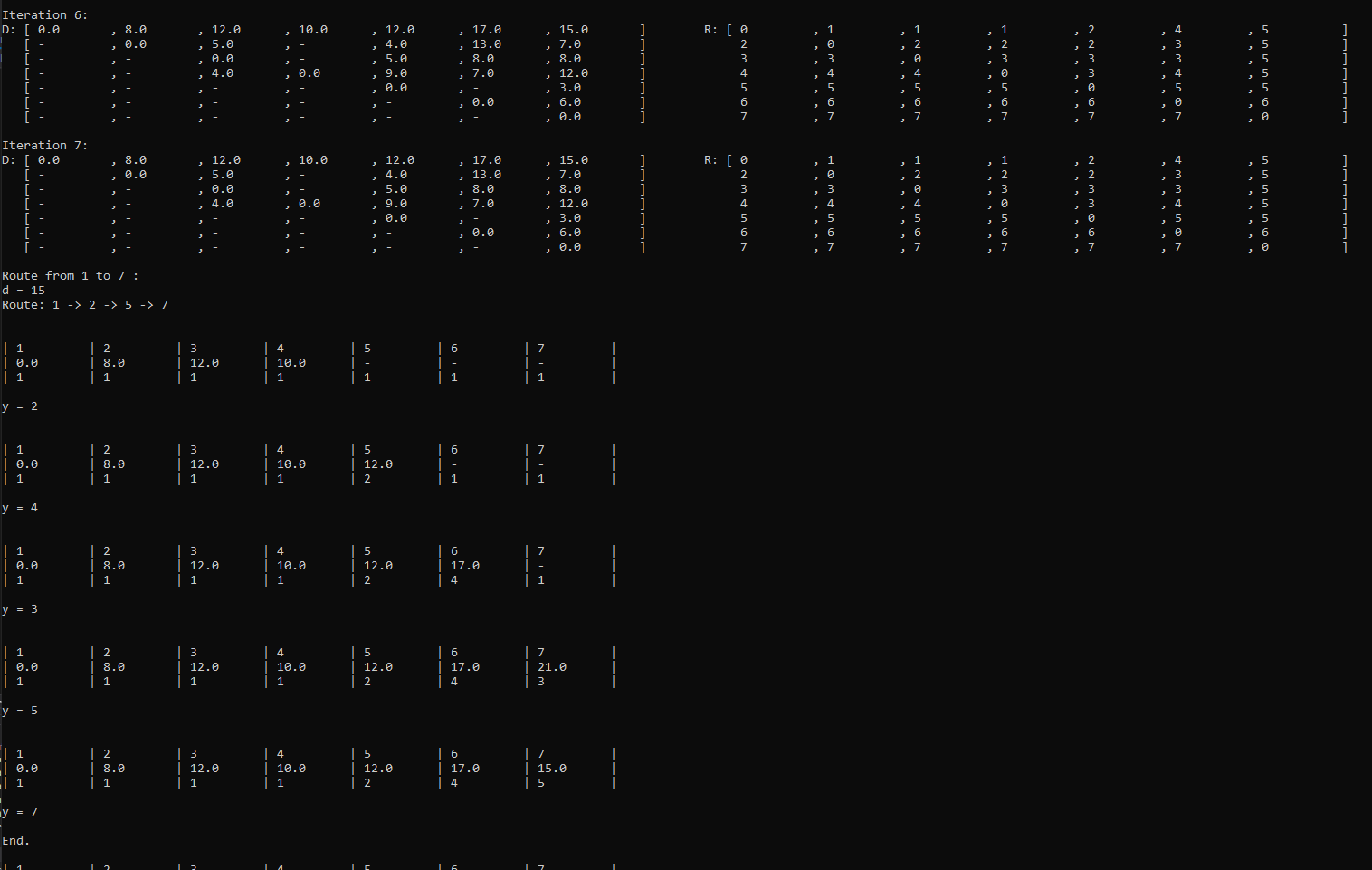


Рис. 3. Результати програми

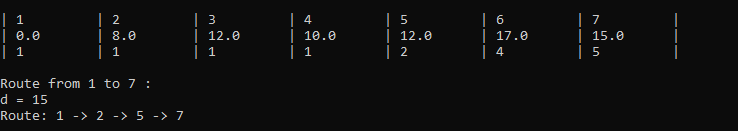


Рис. 4. Результати програми

**ВИСНОВКИ**

На цій лабораторній роботі я ознайомвся на практиці із основними алгоритмами розв’язування потокових задач, навчився знаходити оптимальні маршрути між вершинами мережі за допомогою модифікованих алгоритмів Дейкстри та Флойда.